**TP : Implémentation de CQRS et Axon dans un Microservice avec MySQL**

**Objectifs**

1. Comprendre le concept de **CQRS (Command Query Responsibility Segregation)**.
2. Mettre en place un microservice avec **Axon Framework**. (axon server dans un conteneur Docker)
3. Utiliser **MySQL** comme base de données pour stocker les événements et les entités.
4. Développer un microservice simple de gestion de produits.

Pour ajouter CQRS (Command Query Responsibility Segregation) et Axon à votre microservice product tout en utilisant MySQL comme base de données, voici les étapes principales :

---

**Étape 1 : Ajouter les dépendances nécessaires**

Modifiez le fichier pom.xml pour inclure les dépendances Axon et Spring Boot Starter. Ajoutez également la dépendance pour MySQL.

```
<dependencies>
  <!-- Axon Framework -->
  <dependency>
    <groupId>org.axonframework</groupId>
    <artifactId>axon-spring-boot-starter</artifactId>
    <version>4.8.0</version>
  </dependency>
```

---

**Étape 2 : Configurer Axon et la base de données MySQL**

Dans le fichier application.properties ou application.yml, configurez Axon et MySQL.

```
# Configuration de la base de données
spring.datasource.url=jdbc:mysql://localhost:3306/product_microservice
spring.datasource.username=your_username
spring.datasource.password=your_password
spring.jpa.hibernate.ddl-auto=update
```

**# Configuration d'Axon**

**axon.eventhandling.processors.*.mode=tracking**

**axon.eventhandling.processors.*.source=eventStore**

**axon.eventhandling.processors.*.initial-segment-count=1**

**# Utilisation de MySQL pour stocker les événements**

**axon.eventstore.jpa.schema-generation.enabled=true**

---

**Étape 3 : Créer les Commandes, Événements et Modèles**

**Dans un dossier cqrs:**

**Commande pour créer un produit :**

```
import org.axonframework.modelling.command.TargetAggregateIdentifier;


public class CreateProductCommand {
   @TargetAggregateIdentifier
   private final String id;
   private final String name;
   private final String description;
   private final double price;
   private final int stock;


   public CreateProductCommand(String id, String name, String description, double price, int
stock) {
      this.id = id;
      this.name = name;
      this.description = description;
      this.price = price;
      this.stock = stock;
   }
```

```
  // Getters
}
```

**Événement associé :**

```
public class ProductCreatedEvent {
  private final String id;
  private final String name;
  private final String description;
  private final double price;
  private final int stock;

  public ProductCreatedEvent(String id, String name, String description, double price, int stock)
{
    this.id = id;
    this.name = name;
    this.description = description;
    this.price = price;
    this.stock = stock;
  }

  // Getters
}
```

---

**Étape 4 : Créer l'Aggregate**

L'Aggregate est le cœur de la gestion des commandes.

```
import org.axonframework.commandhandling.CommandHandler;
import org.axonframework.eventsourcing.EventSourcingHandler;
import org.axonframework.modelling.command.AggregateIdentifier;
import org.axonframework.spring.stereotype.Aggregate;

@Aggregate
public class ProductAggregate {
```

```
  @AggregateIdentifier
  private String id;
  private String name;
  private String description;
  private double price;
  private int stock;

  public ProductAggregate() {
    // Constructeur sans argument requis par Axon
  }

  @CommandHandler
  public ProductAggregate(CreateProductCommand command) {
    // Publier un événement
    AggregateLifecycle.apply(new ProductCreatedEvent(
        command.getId(),
        command.getName(),
        command.getDescription(),
        command.getPrice(),
        command.getStock()
    ));
  }

  @EventSourcingHandler
  public void on(ProductCreatedEvent event) {
    this.id = event.getId();
    this.name = event.getName();
    this.description = event.getDescription();
    this.price = event.getPrice();
```

```
    this.stock = event.getStock();
  }
}
```

---

**Étape 5 : Créer le Projecteur pour les requêtes**

Utilisez un @QueryHandler pour répondre aux requêtes.

**Query :**

```
public class GetProductByIdQuery {
  private final String id;

  public GetProductByIdQuery(String id) {
    this.id = id;
  }

  // Getter
}
```

**Gestionnaire de requêtes :**

```
import org.axonframework.queryhandling.QueryHandler;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import tn.univ.productmicroservice.entities.Product;
import tn.univ.productmicroservice.repositories.ProductRepository;

@Component
public class ProductProjection {

  @Autowired
  private ProductRepository productRepository;

  @QueryHandler
```

```
  public Product handle(GetProductByIdQuery query) {
    return productRepository.findById(query.getId()).orElse(null);
  }
}
```

---

**Étape 6 : Exposer les Commandes et Requêtes via REST**

Ajoutez des points d'entrée REST pour exécuter les commandes et gérer les requêtes.

**Contrôleur des commandes :**

```
import org.axonframework.commandhandling.gateway.CommandGateway;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/products")
public class ProductCommandController {

  @Autowired
  private CommandGateway commandGateway;

  @PostMapping
  public String createProduct(@RequestBody Product product) {
    String id = UUID.randomUUID().toString();
    CreateProductCommand command = new CreateProductCommand(
        id, product.getName(), product.getDescription(), product.getPrice(),
product.getStock());
    commandGateway.sendAndWait(command);
    return id;
  }
}
```

---

**Étape 7 : Vérifier et Tester**

1. **Créer un produit :** Utilisez l'API POST pour envoyer une commande.
2. **Consulter un produit :** Exécutez une requête via une API GET pour vérifier les données.